# Noise-contrastive estimation of normalising constants and GANs

## Contents

**1 Fonctions génériques** **2**
1.1 Algorithme d'Hasting . . . . . . 2
1.2 MC MLE (Geyer) . . . . . . 3
1.3 NCE (Gutmann) . . . . . . 3
1.4 Graphiques . . . . . . 4

**2 Applications** **6**
2.1 Exemple basique : la loi normale . . . . . . 6

**3 Nouvelles approches** **11**
3.1 Bootstrap . . . . . . 11
3.2 Récursivité . . . . . . 12
3.3 Reverse logistic regression . . . . . . 14

# 1 Fonctions génériques

## 1.1 Algorithme d'Hasting

Utilité : simuler selon $p_m(.,\psi)$ pour un paramètre $\psi$ choisi.

| Argument | Type | Exemple | Indication |
|---|---|---|---|
| x | vecteur | rcauchy(100, 0, 1) | notre échantillon de densité inconnue |
| n | entier | 100 | taille de la simulation |
| psi | vecteur | c(0,1) | paramètres de la fonction h |
| h | fonction | | fonction qui retourne $\overline{p_m}(.,\psi)$ |

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Ci-dessous une autre version qui génère un échantillon iid.

| Argument | Type | Exemple | Indication |
|---|---|---|---|
| x | vecteur | rcauchy(100, 0, 1) | notre échantillon de densité inconnue |
| n | entier | 100 | taille de la simulation |
| psi | vecteur | c(0,1) | paramètres de la fonction h |
| h | fonction | | fonction qui retourne $\overline{p_m}(.,\psi)$ |
| $\epsilon$ | Entier | 2 | pas de décorrélation $\overline{p_m}(.,\psi)$ |

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
```

```
  return (y)
}
```

## 1.2 MC MLE (Geyer)

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, n, psi, h){

  m = length(x)

  y = hasting(x, n, psi, h)

  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
    control = list(fnscale=-1),
    fn = L
  )$par

  return(theta)
}
```

## 1.3 NCE (Gutmann)

Utilité : Retourne l'estimation de la constante et des paramètres.

| Argument | Type | Exemple | Indication |
|---|---|---|---|
| x | vecteur | rcauchy(100, 0, 1) | notre échantillon de densité inconnue |
| law_y | fonction | rnorm | fonction qui retourne un échantillon suivant la loi $p_n$ |
| n | entier | 100 | taille de l'échantillon de bruit suivant la loi $p_n$ |
| params_y | vecteur | c(0,1) | arguments de la fonction law_y |
| log_pm | fonction | | fonction qui retourne le logarithme de la densité $p_m$ |
| log_pn | fonction | | fonction qui retourne le logarithme de la densité $p_n$ |
| size_theta | entier | 3 | taille de $\theta$, vaut habituellement 2 ou 3 |
| method | string | "CG" | méthode d'optimisation, habituellement "CG" ou "BFGS" |

```
nce = function(x, law_y, params_y, log_pm, log_pn, size_theta, n){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }
```

```
  theta = optim(
    par = rep(1, size_theta),
    gr = "CG",
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

### 1.4 Graphiques

Utilité : afficher l'histogramme pour un échantillon de données $x$.

```
print_hist = function(x) {
  df = data.frame(x = x)
  hist_x = ggplot(df, aes(x=x)) +
           geom_histogram(aes(y = stat(count)/sum(count)), bins = 20, color="white") +
           theme(aspect.ratio = 1) +
           labs(y = "Fréquence") +
           ggtitle("Distribution de l'échantillon x")
  print(hist_x)
}
```

Utilité : pour NCE, afficher l'évolution des paramètres au fur et à mesure de l'augmentation de n (la dimension de l'échantillon de bruit)

```
NCE_evol_params = function(x, law_y, params_y, log_pm, log_pn, size_theta, ratio, steps, labels) {

  # Creation de l'abscisse
  m = length(x)
  N = seq(0, m*ratio, length.out = steps + 1)

  # Creation de l'ordonnée
  theta = c()
  for (n in N) {
    theta = append(theta, nce(x, law_y, params_y, log_pm, log_pn, size_theta, n))
  }

  # Formatage des données
  theta = t(rbind(matrix(theta, nrow = size_theta),N))
  df = as.data.frame(theta)
  df_melted = melt(df, id.vars = "N")

  # Plot
  plot_df = ggplot(df_melted, aes(x = N, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  geom_point(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par rapport au bruit",
       x = "n (taille du bruit)",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))
```

```
  print(plot_df)

  #return(theta)
}
```

Note : il faudrait optimiser le temps de calcul de ces fonctions, peut-être en matriciel au lieu des boucles ou bien avec du calcul en parralèle sur CPU/GPU

# 2 Applications

## 2.1 Exemple basique : la loi normale

Soit $x$ l'échantillon de taille $m$ obtenu selon la loi de densité inconnue $p_d$.

On considère ici que $p_d$ appartient à la famille de fonctions paramétrées par $\theta = (c, \mu, \sigma)$ suivante :

$$p_m(u;\theta) = \frac{1}{Z(\mu,\sigma)} \times exp\left[-\frac{1}{2}\left(\frac{u-\mu}{\sigma}\right)^2\right] \quad \text{d'où} \quad ln(p_m(u;\theta)) = c - \frac{1}{2}\left(\frac{u}{\sigma} - \frac{\mu}{\sigma}\right)^2$$

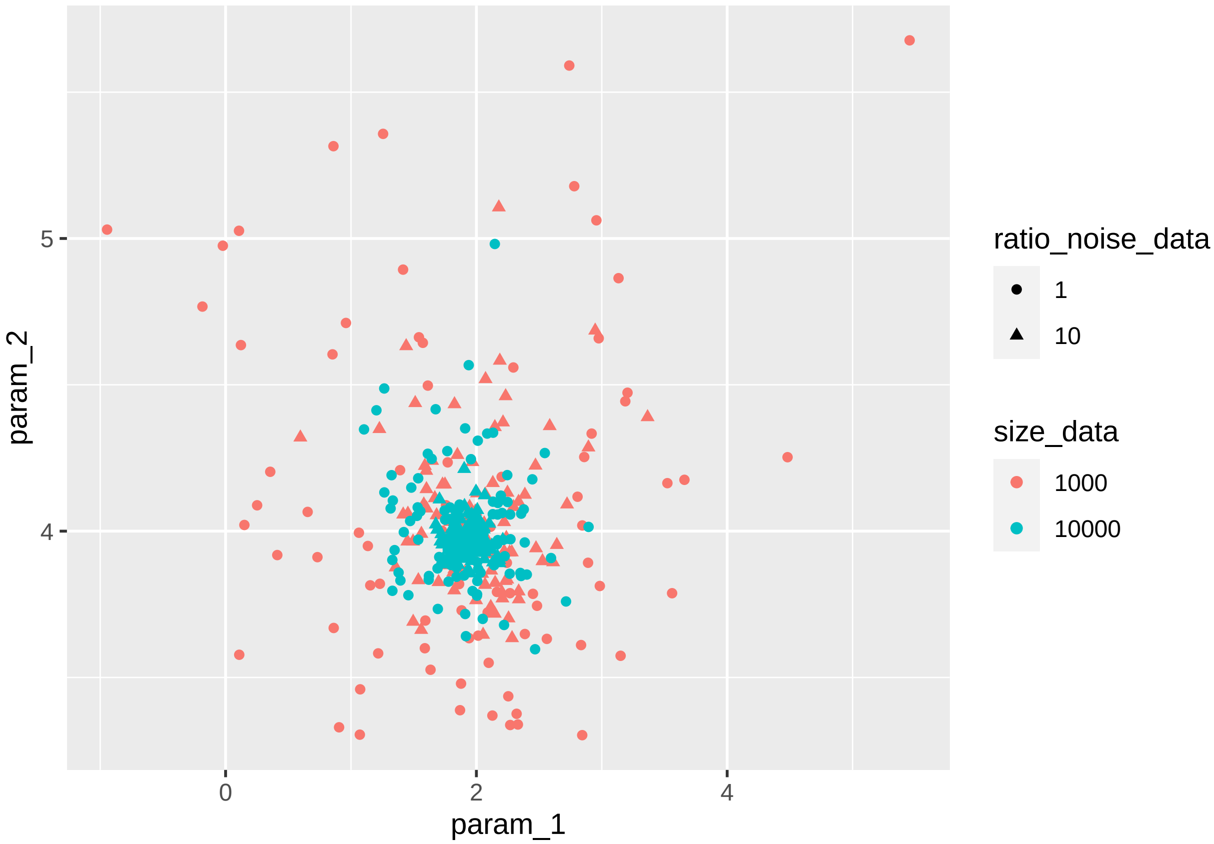
```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
}

log_pm = function(u,theta){
  return(theta[3] - 1/2 * (u/theta[2] - theta[1]/theta[2]) ** 2)
  # theta[1] = mu / theta[2] = sigma / theta[3] = c
}

log_pn_cauchy = function(u){
  return(log(dcauchy(u, mean(x), sd(x))))
}

m = 10000
n = 10000
x = rnorm(m, 2, 4)
size_theta = 3
```

```
# METHODE MC MLE
mc_mle(x, n, c(mean(x),sd(x)), pm_barre)
```

```
## [1] 2.081282 4.120032
```

Etudions l'impact de la dimension des échantillons sur la convergence des estimateurs.

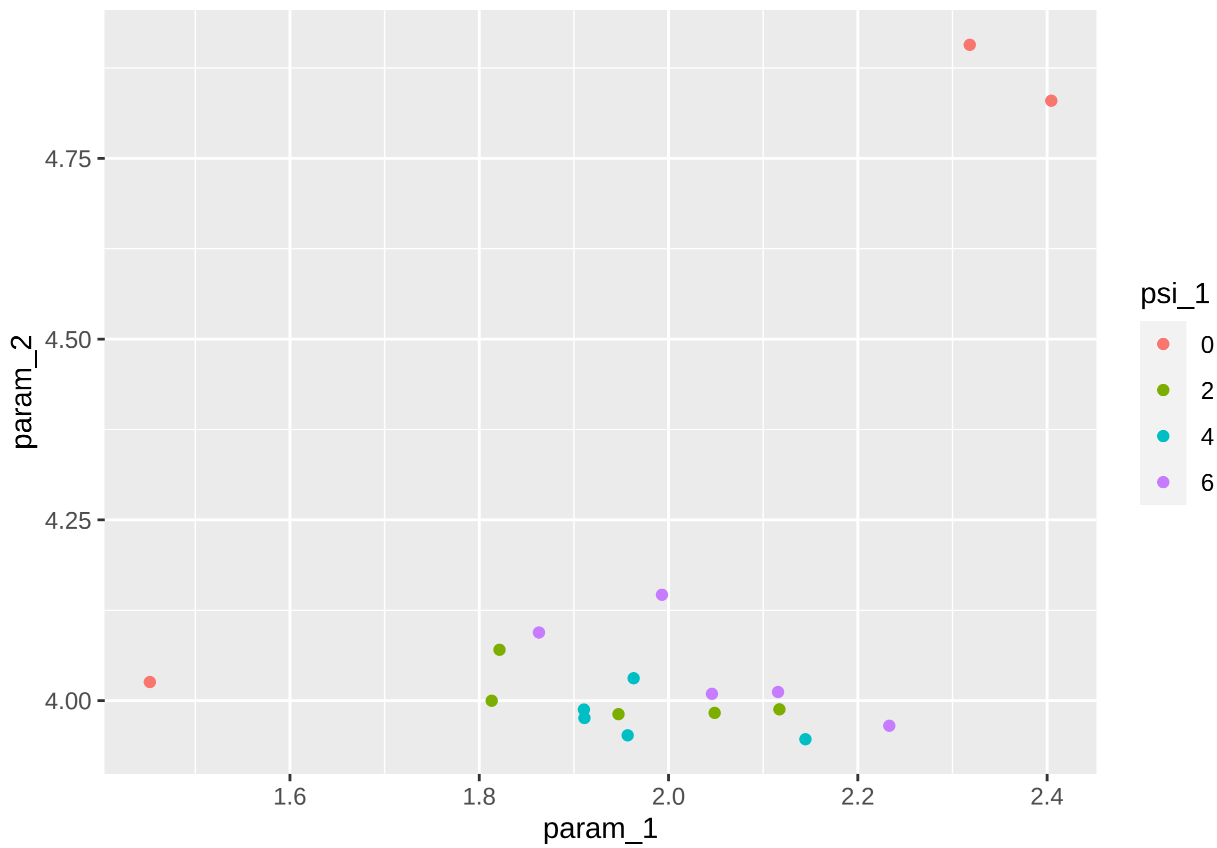
```
ggplot(df_mcmle_filt, aes(x = param_1, y = param_2, color = size_data, shape = ratio_noise_data)) + geom
```




Etudions l'impact du choix de $\psi$ sur la convergence des estimateurs.

```
df_mcmle_psi = data.frame(matrix(ncol = 4, nrow = 0))
colnames(df_mcmle_psi) = c("param_1", "param_2", "psi_1", "psi_2")

psi_1 = c(0, 2, 4, 6)

x = rnorm(m, 2, 4)
for (k in psi_1){
  for (i in 1:5) {
      df_mcmle_psi[nrow(df_mcmle_psi) + 1, ] = c(mc_mle(x, m*10, c(k, k+2), pm_barre), k, k+2)
  }
}
```

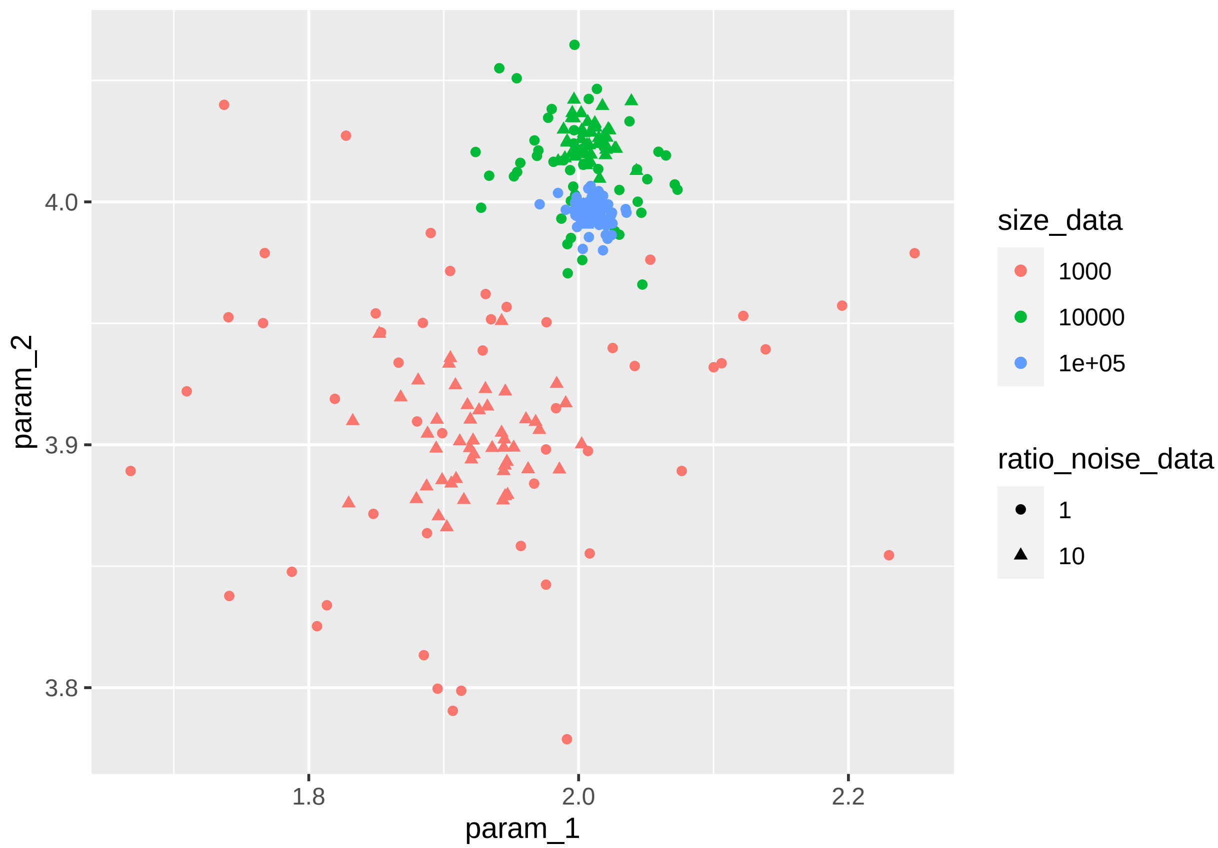
```
df_mcmle_psi$psi_1 = as.factor(df_mcmle_psi$psi_1)
df_mcmle_psi = filter(df_mcmle_psi, param_1 <= 3)
ggplot(df_mcmle_psi, aes(x = param_1, y = param_2, color = psi_1)) + geom_point()
```



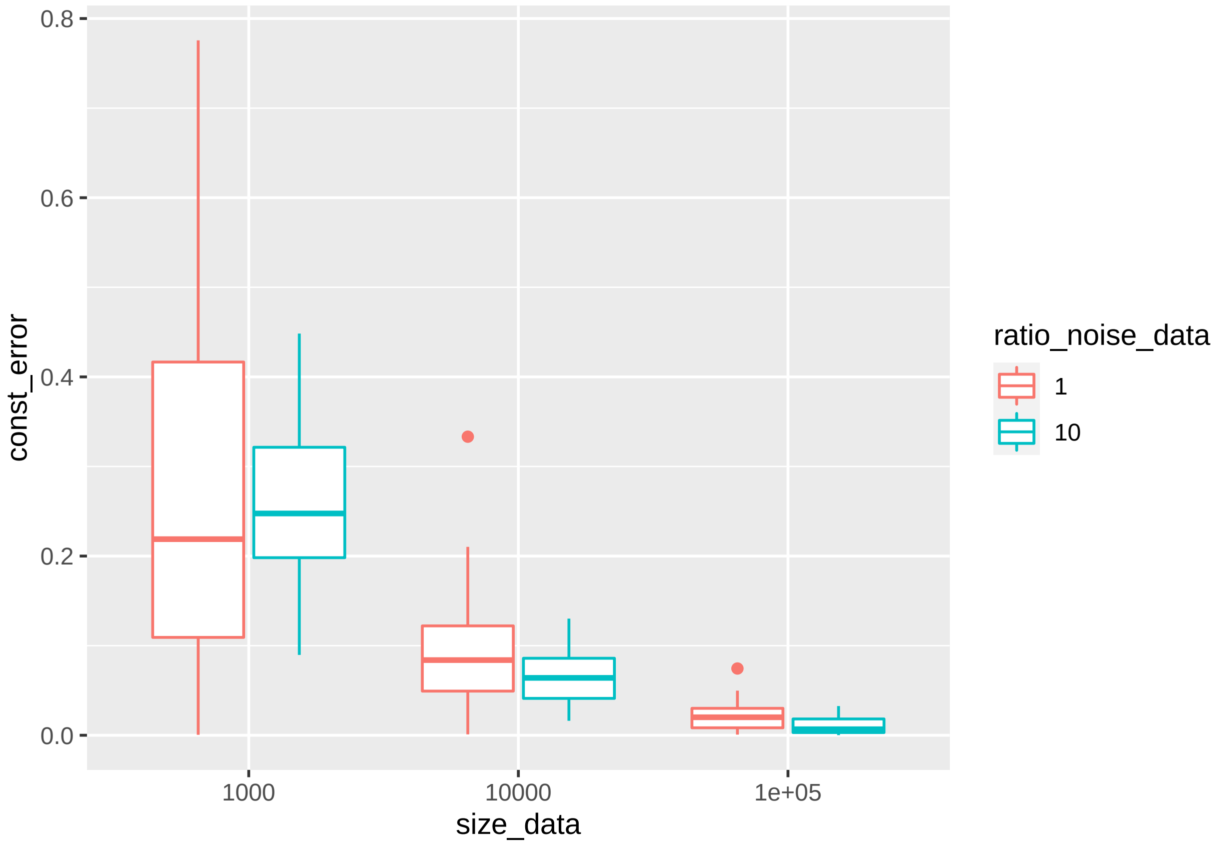

```r
# METHODE NCE
nce(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n)
```

```
## [1] 1.978654 3.965714 9.813705
```

```r
ggplot(df_nce, aes(x = param_1, y = param_2, color = size_data, shape = ratio_noise_data)) + geom_point
```




```
ggplot(df_nce, aes(x = size_data, y = const_error, color = ratio_noise_data)) + geom_boxplot()
```


0.8
0.6
0.4
0.2
0.0
const_error
1000
10000
1e+05
size_data
ratio_noise_data
1
10

# 3 Nouvelles approches

## 3.1 Bootstrap

```
# Calcul de {size_boot} estimateurs par bootstrap
NCE_bootstrap = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_boot, labels) {
  m = length(x)
  theta_bootstrap = c()
  x_bootstrap = x
  for (i in 1:size_boot) {
    theta_bootstrap = append(theta_bootstrap, nce(x_bootstrap,
                                              law_y,
                                              params_y,
                                              log_pm,
                                              log_pn,
                                              size_theta,
                                              n))
    x_bootstrap = sample(x, size = m, replace=TRUE)
  }
  return(matrix(theta_bootstrap, nrow = size_theta))
}

# Plot la moyenne empirique des estimateurs bootstrap en fonction du nombre d'estimateurs
NCE_bootstrap_plot = function(matrix_theta_bootstrap) {

  # Formatage des données pour plot
  array_boot = 1:length(matrix_theta_bootstrap[1,])
  df = as.data.frame(cbind(t(rowCumsums(matrix_theta_bootstrap))/array_boot,array_boot))
  df_melted = melt(df, id.vars = "array_boot")

  # Plot
  plot_df = ggplot(df_melted, aes(x = array_boot, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par bootstrap",
       x = "Taille du bootstrap",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))
  print(plot_df)
}


# etude bootstrap de l'estimateur
bootstrap = function(matrix, alpha){
  return(data.frame(
    theta = matrix_theta_bootstrap[,1],
    biais = rowMeans(matrix_theta_bootstrap) - matrix_theta_bootstrap[,1],
    IC = rowQuantiles(matrix_theta_bootstrap, probs = c(alpha/2, 1-alpha/2))
  ))
}
```

```
x_test = rnorm(1000,2,4)

matrix_theta_bootstrap = NCE_bootstrap(x_test, rcauchy, c(mean(x_test),sd(x_test)), log_pm, log_pn_cauc
```

```
kable(bootstrap(matrix_theta_bootstrap, 0.05))
```

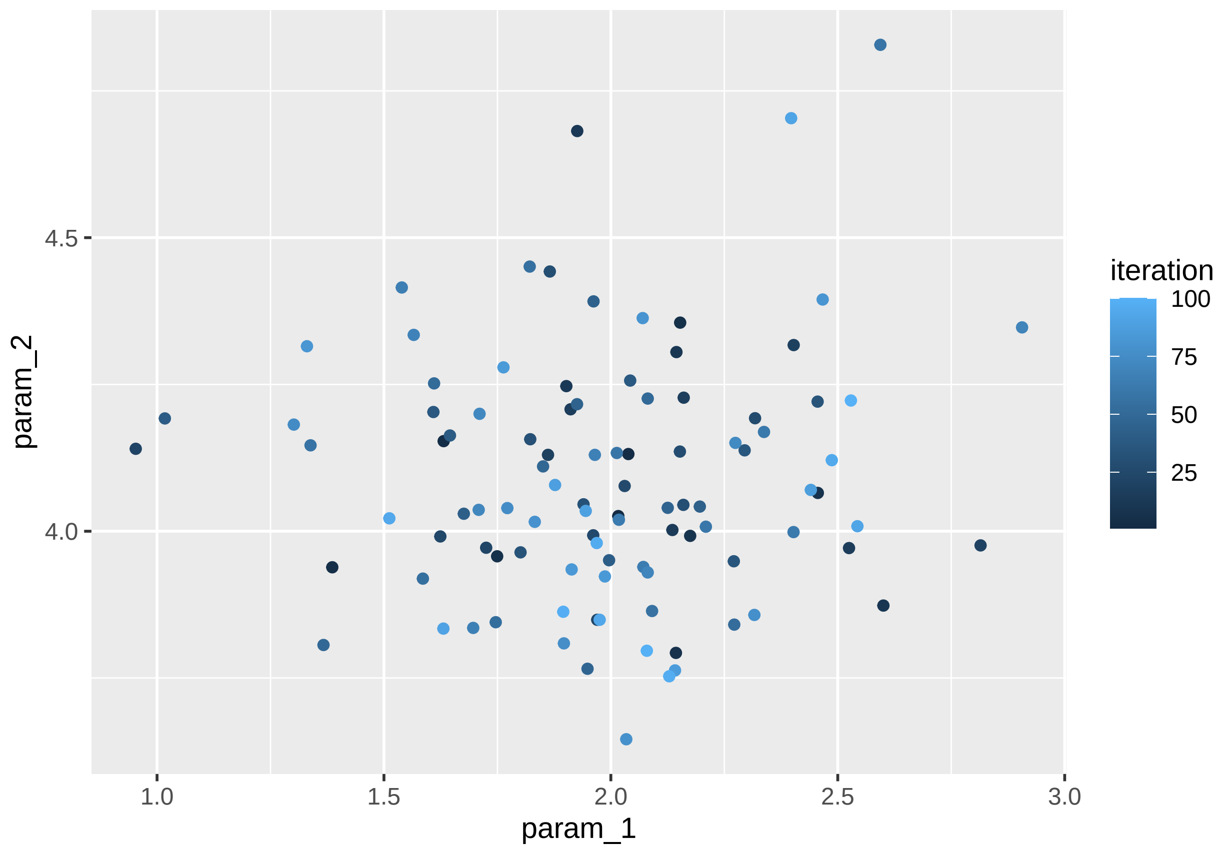
| theta | biais | IC.2.5. | IC.97.5. |
|---:|---:|---:|---:|
| 1.931821 | -0.0318887 | 1.710683 | 2.101345 |
| 3.889665 | 0.0236668 | 3.802154 | 4.079257 |
| 9.996715 | -0.1370528 | 9.468038 | 10.336846 |

### 3.2 Récursivité

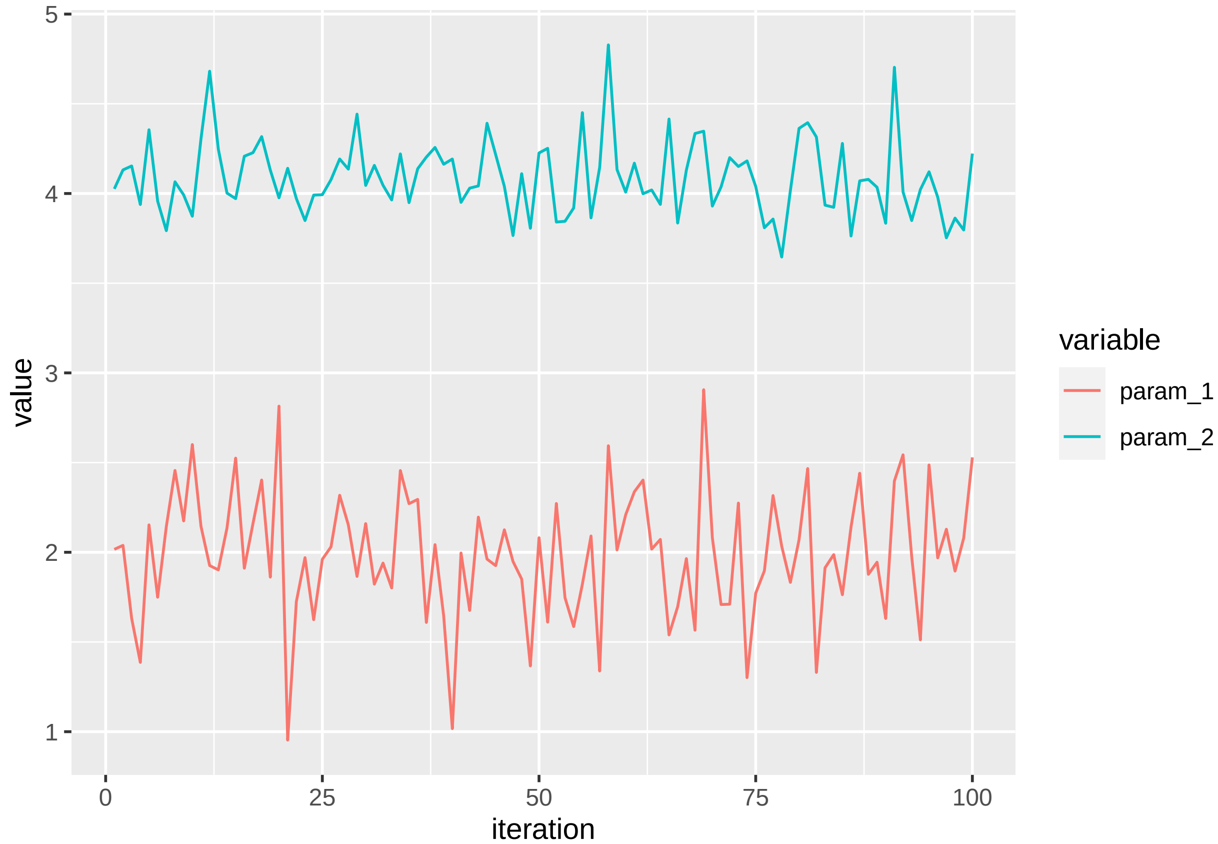
Utilité : améliorer récursivement la précision de l'estimation via les estimations précédentes

```
m = 10000
n = 10000
x = rnorm(m, 2, 4)
psi = c(mean(x),sd(x))

PSI = data.frame(matrix(ncol = 3, nrow = 0))
colnames(PSI) = c("iteration", "param_1", "param_2")

for (i in 1:100) {
  y = hasting(x, n, psi, pm_barre)
  PSI[nrow(PSI) + 1, ] = c(i, psi)

  L = function(theta){
    return(sum(log(pm_barre(x,theta)/pm_barre(x,psi))) - m*log(mean(pm_barre(y,theta)/pm_barre(y,psi)))
  }

  psi = optim(
    par = rep(1,length(psi)),
    gr = "CG",
    control = list(fnscale=-1),
    fn = L
  )$par
}
```

```
ggplot(PSI, aes(x = param_1, y = param_2, color = iteration)) + geom_point()
```




```
PSI_melted = melt(PSI, id.vars = "iteration")
ggplot(PSI_melted, aes(x = iteration, y = value)) + geom_line(aes(color = variable, group = variable))
```




Cette approche n'améliore pas la précision de notre estimation.

### 3.3 Reverse logistic regression

La maximisation de la fonction objectif

$$l_n(\eta) = \sum_{j=1}^{m}\sum_{i=1}^{n_j} log(p_j(X_{i,j},\eta))$$

permet d'estimer les $\eta$ (qui sont fonction des constantes de normalisation des $h_j$). On utilise les notations suivantes :

$$\eta_j = -log(Z_j) + log(\frac{n_j}{n}) \text{ avec } Z_j \text{ la constante de normalisation de } h_j$$

$$p_j(x) = \frac{h_j(x)e^{\eta_j}}{\sum_{k=1}^{m} h_k(x)e^{\eta_k}}$$

Exemple avec $m = 2$, $n = n_1 + n_2 = 1000 + 1000$, et pour coller avec les méthodes différentes on va prendre $h_1$ la densité non normalisée d'une $\mathcal{N}(\alpha)$ dont on a estimé $\alpha$ par MC MLE et $h_2$ la densité non normalisée d'une $\mathcal{N}(\psi)$ avec $\psi$ qu'on choisit.

```
rev_log_reg = function(x, alpha, n, psi, h){

  m = length(x)
  y = hasting(x, n, psi, h)

  # calcul des probabilités p_j
```

```
  denom = function(sample, eta) {
    return(pm_barre(sample, alpha)*exp(eta[1]) + pm_barre(sample,psi)*exp(eta[2]))}
  p_1 = function(sample, eta){
    return (pm_barre(sample, alpha)*exp(eta[1]) / denom(sample, eta))}
  p_2 = function(sample, eta){
    return (pm_barre(sample, psi)*exp(eta[2]) / denom(sample, eta))}

  # fonction objectif
  L = function(eta) {
    return(sum(log(p_1(x, eta)) + sum(log(p_2(y, eta))))}

  # initialisation descente de gradient
  eta1 = -log(sd(x)*sqrt(2*pi)) + log(m/(m+n))
  eta2 = -log(sd(y)*sqrt(2*pi)) + log(n/(m+n))

  # optimisation
  const = optim(
    par = c(eta1,eta2),
    gr = "CG",
    control = list(fnscale=-1),
    fn = L
  )$par

  a = exp(-const[1] + log(m/(m+n)))
  b = exp(-const[2] + log(n/(m+n)))
  return(c(a,b))
}
```

```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
}

m = 10000
n = 100000
x = rnorm(m,2,4)
psi = c(mean(x), sd(x))

alpha = mc_mle(x, n, psi, pm_barre)
print(alpha)
```

```
## [1] 1.957965 4.112372
```

```
print(rev_log_reg(x, alpha, n, c(8,8), pm_barre))
```

```
## [1] 10.10388 18.76937
```

```
print(rev_log_reg(x, alpha, n, c(5,2), pm_barre))
```

```
## [1] 9.957133 4.789113
```

```
print(rev_log_reg(x, alpha, n, c(2,4), pm_barre))
```

```
## [1] 9.564931 9.321253
```